RJSSIM: A RECONFIGURABLE JOB SCHEDULING SIMULATOR FOR PARALLEL PROCESSING LEARNING

Luís F. W. Góes¹, Carlos A. P. S. Martins²

Abstract - In this work, we present a job scheduling simulator called RJSSim, used as an aid tool for parallel processing learning. This software is a functional and performance discrete-event Java-based simulator of job scheduling policies. Our objectives are: to propose and implement RJSSim simulator; to verify the functioning of RJSSim; to show the use of RJSSim for parallel processing learning; to compare and analyze the characteristics of some related job scheduling simulators; to present the reconfigurability concept and its application on job scheduling policies. We analyzed characteristics of some job scheduling simulators. Then we proposed and implemented a RJSSim prototype and verified its functioning using some Bounded Number of Processors (BNP) scheduling algorithms. After, we implemented a reconfigurable BNP scheduling policy and compared with the other ones. Finally, we analyzed the use of RJSSim as an aid tool for parallel processing learning. Our main contributions are: an implementation of RJSSim as a learning aid tool; a characteristic analysis of related simulators; an implementation of a reconfigurable job scheduling policy.

Index Terms - Discrete-event simulation, Java-based simulator, Parallel processing learning, Reconfigurable job scheduling.

INTRODUCTION

There are three well-known performance evaluation techniques: measurement, analytical modeling and simulation [8]. Measurement always needs a real environment, so a grid or cluster environment can be very expensive, or cannot exist. Analytical modeling requires too much simplifications and assumptions, besides it needs a good experience on modeling. Complex real-world systems with stochastic elements can be expensive and cannot be accurately described by a mathematical model, then simulation appear as a less expensive (cost) alternative than measurement and more accurate than analytical modeling.

In simulation, we use a computer to evaluate a model numerically and data are gathered in order to estimate the desired true characteristics of the model. It allows a more detailed, flexible and controlled environment. So, analysts can compare some alternatives under a wide variety of workloads [2][8][10]. Further, there are many available tools that aid in simulations development: simulation libraries (SimJava [11][12] etc.), languages (SIMSCRIPT [10], SLAM [10] etc.) and application specific simulators (GridSim [2], Simgrid [3], SRGSim [1] etc.).

Among those tools, simulation libraries are the most portable (use a general-purpose language) and flexible approach. The developer has to learn only a little set of routines to build his simulation, so libraries are easy to implement and use. The main disadvantage is that developers don’t have compiler check, analysis and optimization [8][10][11].

There are four most common types of simulations: Emulation, Monte Carlo, Trace-Driven and Discrete-Event simulation [8][10]. Discrete-Event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. These points are the ones at which an event occurs, where an event is defined as an instantaneous occurrence that may change the system’s state [2][8][10].

Scheduling is the decision of which task will execute on a certain processing element at a certain time [6][7]. The objective of scheduling is to minimize the completion time of a job by properly allocating the tasks to the processors and sequencing their execution [9]. There are various scheduling classifications: dynamic scheduling, static scheduling, load balancing, preemptive scheduling, non-preemptive scheduling, etc. Scheduling policies schedule jobs and tasks according to some priorities, resources availability, performance metrics and user type. They have to be highly configurable to attend the complex needs of the users [7][9].

Reconfigurable computing is a new and revolutionary paradigm. Its concepts can increase systems flexibility, adaptability and performance. In many works [4][5][16], the word reconfigurable has been used as a synonym for flexible, adaptable and compositional. As presented in [13], we relate the word reconfigurable to a compositional aspect. Thus, a reconfigurable system is a system composed by interconnected constructive blocks. The system’s behavior is defined by the interconnection topology among constructive blocks and these ones functioning logic. So, the system’s behavior can change dynamically according to its form (space connection) [5]. As a consequence, the system becomes more flexible and adaptable.

¹ Luís F. W. Góes, Pontifical Catholic University of Minas Gerais, Post-Graduation Program in Electrical Engineering, Computational and Digital Systems Laboratory, Belo Horizonte, MG, Brazil lwg@mailbr.com.br
² Carlos A. P. S. Martins, Pontifical Catholic University of Minas Gerais, Computer Science Department and Post-Graduation Program in Electrical Engineering, Computational and Digital Systems Laboratory, Belo Horizonte, MG, Brazil capsman@pucminas.br
Nowadays, parallel processing disciplines are commonly offered in Computer Science and Electrical Engineering courses. The infrastructure necessary to teach these courses is very expensive. Because of economical problems, many universities from developing countries do not offer undergraduate parallel processing courses. So, some works [15] suggest alternatives to this problem like the use of clusters, free softwares etc. In this work, we believe that simulation is a cheap and highly didactic alternative to this problem.

The motivations for this work are: simulation can reduce problems (infrastructure etc.) to learn and experiment parallel processing concepts in practice; reconfigurability concepts can increase flexibility, adaptability and performance of job scheduling policies.

The objectives of this work are: to propose and implement RJSSim simulator; to verify the functioning of RJSSim; to show the use of RJSSim for parallel processing learning; to compare and analyze the characteristics of some related job scheduling simulators; to present the reconfigurability concept and its application on job scheduling policies. The goals are: an implementation of RJSSim prototype as a learning aid tool; a characteristic analysis of related simulators; an implementation of a reconfigurable job scheduling policy.

METHOD

The method used to reach the objectives is divided in the following six steps: I) Comparison and analysis of related job scheduling simulator; II) Purpose, implementation and presentation of RJSSim; III) Verification of RJSSim’s functioning for the Bounded Number of Processors (BNP) scheduling algorithms, comparing with the manual method; IV) Design, implementation and verification of a reconfigurable scheduling policy for this class of algorithms; V) Functional and performance analysis of the implemented scheduling policies; VI) Presentation of the RJSSim’s usage for parallel processing learning. Those steps will be described in the next sections.

RELATED JOB SCHEDULING SIMULATORS

Job scheduling is a classic example of computer simulation. So, there are so many job scheduling simulators of sequential, parallel and distributed machines [1][2][10]. Among various job scheduling simulators we will analyze GridSim, Simgrid and SRGSim (Shark Group Simulator). In spite of having different purposes, their characteristics combination generated the RJSSim simulator. GridSim is a Java based framework developed in Monash University at Melbourne. It supports modeling and discrete-event simulation of heterogeneous resources, such as single or multiprocessors, shared and distributed memory machines, and job scheduling on parallel and distributed systems (clusters, grids and P2P systems). The main characteristics of GridSim are: time and space shared resources modeling, support of various parallel application models, CPU and I/O intensive jobs, grid information service, distributed users and scheduling and high portability. The main disadvantages are: network parameters like latency, throughput and topology cannot be specified, job parameters described only as constants (can not use traces or probabilistic models) and jobs are not communication bounded [2].

Simgrid is a C++ based framework developed in the University of California at San Diego. It simulates job scheduling in time-sharing resources, in which a workload can be represented by constants or traces. DAGSim is a simulator build on top of Simgrid that allows jobs representation (DAG files) and architecture (Grid files). The main characteristics of Simgrid / DAGSim are: DAG scheduling, resources can be described by two metrics (latency and service rate), various network topologies are simulated, prediction with arbitrary error behavior and trace use. The main disadvantages are: jobs are only CPU-bound, simulation of only time-shared systems, support of only one scheduling entity, use of simple statistical analysis mechanisms and text interface [3].

SRGSim is a Java based discrete event simulation framework developed in the University of California. It simulates various classes of job scheduling (dynamic, static, load balancing), resources (clusters, MPPs etc.) and jobs through probabilistic models, DAGs and constants. The main characteristics of SRGSim are: provides a DAG editor, jobs are described by a simulation language, traces, DAGs or probabilistic models, jobs are CPU, I/O and Communication bounded, various network topologies are simulated and parallel implementation using sockets. The main disadvantages are: simulations of only time-shared systems, text interface and grids are not supported [1].

We can observe that all simulators do not have a GUI and do not permit multi-level scheduling (node, cluster and grid schedulers). Only SRGSim uses probabilistic models and supports communication-bounded jobs. But GridSim is the only one that allows multiple users submitting jobs. Simgrid provides DAG scheduling and the use of traces. So, we conclude that the combination of all these characteristics could compose one single job scheduling simulator tool.

RJSSIM PRESENTATION

RJSSim is a reconfigurable job scheduling simulator. This reconfigurable software is a functional and performance discrete-event Java-based simulator of job scheduling policies. RJSSIm uses dynamic simulation (a model represents a system as it evolves over time), stochastic simulation model (contains some probabilistic and random input components) and discrete simulation model. These features compose a discrete-event simulation model.

In Figure 1, we see the RJSSim’s high-level architecture. It is composed by four main independent components: graphical user interface (GUI), entities, core and statistical
module. The **GUI** provides CAD tools for the configuration of jobs, users, schedulers and architectures, simulation setup, statistical and performance information. In the first version, it is composed by seven editors (task, job, simulation, scheduler, user, node and cluster). Those editors facilitate the creation and execution of a simulation environment. The **entities** are the simulation objects and they can be users, schedulers and architectures. They communicate with each other through the RJSSim’s core. The **core** implements a parallel discrete-event simulation, where each entity is an independent thread. It provides an identifier generator and an API to manipulate entities and events. The **statistical module** is responsible for get some data from entities and generate performance and statistical information like response time, efficiency, means etc.

---

**FIGURE 1**

THE HIGH-LEVEL RJSSIM’S ARCHITECTURE

The RJSSim’s core is a general-purpose discrete-event simulation library. It implements the next-time advance mechanism, in which the simulation clock is advanced to the time of the most imminent future event in the queue. The main advantages of RJSSim’s core are: i) simWait() calls are not necessary, because the entity’s body ( ) method automatically receives events from the simulator, until the simulation is over; ii) simHold() calls are not necessary; iii) The events queue is sorted by clock time, so entities that don’t interact with (depend of) other entities can schedule all their events in the same moment; iv) An event can have more than one destination entities; v) The RJSSim provides a standard method to program the simulation. The body ( ) method is composed by a switch case structure in which the developer uses only simSchedule ( ) and simCancel ( ) methods to program the entities behavior.

The simulation is composed by entities that communicate with each other only through events. There are two main entities: SimUser and SimArchitecture (Figure 2). The **SimUser** represents a real user that submits jobs to the architecture. It has a list of jobs that are submitted using the Monte Carlo method and a job inter-arrival probability distribution. Users are linked to the architecture that they will submit jobs. The **SimArchitecture** represents a real architecture (multiprocessor, node, cluster, grid etc.), so each one has its own scheduler. In this prototype, there are only SimNode and SimCluster architectures. But this software architecture design allows n levels of architectures and scheduling. The SimNode has an I/O module and processors, and the SimCluster has a network and nodes.

---

**FIGURE 2**

THE RJSSIM’S CLASS HIERARCHY DIAGRAM

In RJSSim, a job can be represented by a graph. The nodes represent tasks that can be CPU, I/O and Communication-bounded. Each task is composed by instruction blocks, which in turn have instructions. There are four types of instructions: CPU, I/O, send (non-blocking) and receive (blocking). The CPU instruction represents millions of whetstone (float-pointing) instructions that have to be executed by the processor. The I/O instruction represents the number of Kbytes read and/or write on a disk that is processed by the I/O module. An I/O instruction implies on a task blocking. The receive instruction blocks a task until it receives n messages. The send instruction can have n destination tasks and the message size. With send and receive instructions is possible to simulate group communication operations (broadcast, barrier etc.).

All instructions attributes (message size, number of instructions, Kbytes etc.) and the repetitions of an instruction block are represented by probability distributions. This feature provides a way to put some level of variation and uncertainty in the simulation. It possibilities the creation of a real environment, which rarely has ideal conditions. In Figure 3, we see the RJSSim’s task editor where the user can include instruction blocks, instructions and their attributes.

Schedulers have to extend the SimScheduler class, which has a main method called schedule ( ). This method is called every time an entity receives an event and generally it contains the scheduling policy implementation. A node scheduler has to implement four event methods: jobArrivalEvent (job), taskReadyEvent (task), taskFinishedEvent (task) and endOfQuantumEvent (task). A job arrival event can be scheduled by a user or architecture. A task ready event is generated every time a task receives all necessary messages or read and/or write on a disk that is processed by the I/O module. An I/O instruction implies on a task blocking. The receive instruction blocks a task until it receives n messages. The send instruction can have n destination tasks and the message size. With send and receive instructions is possible to simulate group communication operations (broadcast, barrier etc.).

---

**Session**
After execute all instructions, a task finished event is scheduled. A processor has a quantum to execute a task, if a task is blocked for any reason or the quantum ends, an end of quantum event is generated.

A cluster scheduler has to implement three event methods: jobArrivalEvent (job), jobFinishedEvent (job) and endOfQuantumEvent (task). A job finished event is scheduled when all tasks of a job finished. There are other internal events like taskMessageReception (task) that is responsible to inform the task a message arrival.

**FIGURE 3**

RJSSim’s Task Editor Screen

On each simulation execution is possible to specify a simulation seed (to be used with the distributions) and a maximum clock time to the simulation end. If a clock time is not specified, the simulation ends when all events are scheduled. At the end of the simulation, a log file is generated. This log contains all the events, messages, instructions and other actions executed.

In this first implementation of RJSSim, there are some limitations: supports only node and cluster architectures; dynamic scheduling and load balancing policies are not implemented; a task cannot spawn tasks, networks are described only by their speed; the GUI allows a user per architecture; the statistical module is limited.

**RJSSim Verification**

In this phase, we chose DAG scheduling policies to verify the functioning of RJSSim. This algorithm class is easy to execute using the manual method for graphs with few nodes. So, it possibilities an easy way to verify and compare the results generated by RJSSim.

In the DAG Scheduling Problem, a parallel program can be represented by a directed acyclic graph (DAG) \( G=(V,E) \), where \( V \) is a set of \( v \) nodes and \( E \) is a set of \( e \) directed edges. A node represents a task, which in turn is a set of instructions which must be executed sequentially without preemption in the same processor. The weight of a node \( n_i \) is called the computation cost and is denoted by \( w(n_i) \). The edges correspond to the communication messages (communication cost) and precedence constraints [6][9].

The precedence constraints of a DAG dictate that a node can not start execution before it gathers all of the messages from its parent nodes. The communication cost between two tasks assigned to the same processor is assumed to be zero. The objective of scheduling is to minimize the schedule length (highest finish time among tasks) by proper tasks allocation to the processors and arrangement of tasks execution sequencing [6][9].

**FIGURE 4**

A Simple Graph Example

In general, the DAG scheduling problem is an NP-complete problem. Most scheduling algorithms are based on the classic list scheduling technique. The basic idea of list scheduling is to make a task list by assigning them some priorities, and then repeatedly remove the first node from the list and allocate the node to a processor which allows the earliest start time, until all the nodes are scheduled [6][9].

In our work, we implemented three Bounded Number of Processors (BNP) class scheduling algorithms that are based on list scheduling technique. The three algorithms are: Insertion Scheduling Heuristic (ISH), Modified Critical Path (MCP) and Earliest Time First (ETF). These algorithms are based on two priorities: b-level (length of a longest path) and ALAP (as late as possible start time of a node). More information about those algorithms can be found in [9]. We created a homogenous cluster environment composed by three nodes with FIFO schedulers and used a simple graph example (Figure 4) to verify RJSSim’s functioning.

Using RJSSim, the schedule length for each algorithm was: ISH = 22; MCP = 22; ETF = 20. The results were the same using the manual method. Then we conclude that RJSSim was verified for this class of algorithms.

**Reconfigurable Job Scheduling Policy**

In this work, we consider a reconfigurable software as a software that has at least one reconfigurable layer. This layer’s behavior is defined by the interconnection of constructive blocks belonging to a lower layer.

To model, design and verify reconfigurable softwares we developed a method using object-oriented, component-oriented and port-based objects concepts. Object-oriented paradigm provides polymorphism, data encapsulation, inheritance and extensibility [14]. Component-based paradigm emphasizes composition over programming, so it is possible to reuse classes without programming new ones [14]. Port-based objects are data-oriented, unaware of source inputs and destination outputs and communicate only
between ports [16]. All these features together provide a good mechanism to build reconfigurable softwares.

After the verification of RJSSim for BNP algorithms, we used some steps of our method to develop a first reconfigurable scheduling policy implementation of this algorithm class. We executed the following steps: listed all functionalities needed (reconfigurable layer); identified input and output data; defined constructive blocks aggregating some functionalities (lower layer); defined the programmability level of each block; verified if the model was capable to represent the class of problem; implemented, tested and verified the reconfigurable scheduling policy. The implementation was capable to execute the three BNP algorithms, presenting the same results.

In this implementation, a reconfigurable system has two main components: constructive blocks and linkers. A constructive block is a component that implements a set of similar functionalities. It is composed by input ports, a selection method and output ports. First, a configuration flag sets the method that will be used. Then the input data is loaded on the input ports. After, they are processed by the selected method and the results are written in the output ports. Finally, the linkers connected to the constructive block are activated. A linker is responsible for activate and copy data from a constructive block port to another one. A set of linkers compose the reconfigurable system topology. To execute the system, an input block is activated, generating a cascade effect until data reaches the output block. After execution, the output data is read and the system can be reconfigured, setting the flags and changing the topology.

The reconfigurable scheduling policy is composed by four main constructive blocks: Metric Calculator (MC), ListGenerator (LG), Scheduler and ListUpdater (LU). The MC receives a job as an input and calculates b-level or ALAP metrics. LG generates a list according to the metrics, in descending or ascending order, or a list with the entry tasks of a job. A task list is passed to the Scheduler as an input and it schedules the first or the most appropriate task to the node that gives the earliest start time. Finally, the LU can remove a task or update the list with new ready tasks.

**EXPERIMENTAL RESULTS**

To analyze the performance of the implemented scheduling policies and the use of reconfigurability, we created a more complex example graph, as the workload, and configured a new scheduling policy.

In Figure 5, we see the configuration of a modified MCP algorithm called Reconfigurable-MCP. In the original version, MCP schedules the first task on the list to the processor that allows the earliest time execution. The R-MCP looks for the best pair task-processor in the list to schedule. The linkers are represented by the arrows. The blue arrows indicate the main stream (causes block activation) and the black ones indicate the secondary streams. A linker connects two or more ports from different blocks. Near the linkers we see the data stream that is transmitted between ports. Inside the blocks, we see the configuration flags that select the actual functionality of the block. Input and output blocks are flag-independent. The output blocks function as a conditional block, that is, the execution stops only when all the tasks are scheduled. Otherwise it sends the tasks list back to the scheduler.

In Figure 6, we see the example graph that was created through RJSSim’s GUI. We created the same experimental setup used in the verification phase. But we vary the number of processors from 1 to 5.

In Table I, we observe that ETF algorithm presented better results for the example graph. The maximum speedup achieved was 1.72. The R-MCP algorithm showed a better schedule length over MCP using two processors. This result was achieved because R-MCP uses some optimized mechanisms from ETF.

**TABLE I**

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Processors</th>
<th>ETF</th>
<th>ISH</th>
<th>MCP</th>
<th>R-MCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETF</td>
<td>1</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>ETF</td>
<td>2</td>
<td>32</td>
<td>37</td>
<td>37</td>
<td>32</td>
</tr>
<tr>
<td>ETF</td>
<td>3</td>
<td>31</td>
<td>37</td>
<td>37</td>
<td>32</td>
</tr>
<tr>
<td>ETF</td>
<td>4</td>
<td>29</td>
<td>29</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>ETF</td>
<td>5</td>
<td>29</td>
<td>29</td>
<td>29</td>
<td>38</td>
</tr>
</tbody>
</table>

In this work, we are not looking for the best configuration to the example graph. We intend to show that a certain configuration can be the best one for a specific graph and number of processors. Without reconfigurability concepts, the system is limited to a finite number of possibilities. In this case, the system would be limited to the
three implemented algorithms (ETF, MCP and ISH). The reconfigurability usage provides (ideally) an infinity number of configurations. Our reconfigurable implementation increased the flexibility of the BNP scheduling policy class. So it was possible to mix features from different algorithms into one, improving performance in some cases and making the system more adaptable.

This first version of the reconfigurable scheduling policy can be improved on various aspects: reduce the number of constructive blocks; make the constructive block’s inputs and outputs more compatible; implement new functionalities to the block’s logics; make successive refinements using the developed method. Another important improvement is the use of some artificial intelligence mechanisms to automate the search for an optimal configuration for a situation.

**RJSSim for Parallel Processing Learning**

The RJSSim can be used as an aid tool for the learning of parallel processing and related disciplines (operating systems, computer networks, etc.). It provides mechanisms to test in practice parallel architectures, algorithms, programming models, job scheduling policies etc. Through simple performance metrics, users can analyze the performance of a whole parallel system and its individual parts. The use of probabilistic models shows to users that depending on the used probabilistic distributions, they can get more accurate and reliable results.

Through reconfigurable policies, users can mount or configure their own scheduling policies. This can promote some competition between users (students) to find the best policy for a specific workload. The code of RJSSim is highly extensible, so users can modify and include some classes and study new scheduling algorithms, architectures etc.

In our simple simulation example in the experimental results section, users can observe and learn: some performance metrics like speedup, scalability, level of parallelism and response time (schedule length); the system’s functioning through the log file; advantages of reconfigurability usage etc. This example is limited, because a user submits a single job at time, doesn’t use probability distributions, tasks are not I/O-bounded, only static scheduling etc. So, this tool provides much more resources than the ones showed in this example. The use of all resources provides mechanisms to users see the functioning and influence of each parallel system’s component.

**Conclusion**

Simulation is a cheap and didactic alternative to teach parallel processing disciplines in universities from developing countries. In this work, we presented the RJSSim job scheduling simulator and show how it can be used for parallel processing learning. Some related simulators were compared, and then RJSSim was proposed, implemented and verified. We introduced the reconfigurability concept in job scheduling policies and showed its advantages.

The main contributions are: an implementation of RJSSim as a learning aid tool; a characteristic analysis of related simulators; an implementation of a reconfigurable job scheduling policy.

As future works we remark: new functionalities implementation (network, scheduling policies, architectures etc.), the application of quantitative and qualitative exams on students to evaluate the use of RJSSim in a parallel processing discipline; a comparative analysis between scheduling policies and reconfigurable ones; improvement of the statistical module.

We would like to thank the Post-Graduation Program in Electrical Engineering, Computational and Digital Systems Laboratory (LSDC), CAPES and ProPPG for the support.

**References**


